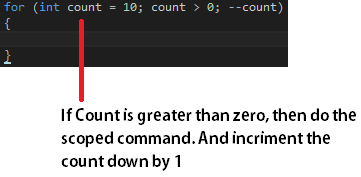
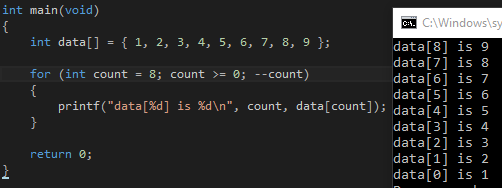


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Entry ID:** | **Date:** | **Day:** | **Start Time:** | **Duration:** | **Session Type:** | **Lab/Lecture Number:** | **Location:** |
| Week 5 | | | | | | | |
| 1 | 27/03/2017 | Monday | 9:00am | 1 hour | Lecture | 13 | WG403 |
| 2 | 27/03/2017 | Monday | 10:00am | 3 hours | Lab |  | WT201 |
| 3 | 30/03/217 | Thursday | 8:00am | 1 hour | Lecture | 14 | WG403 |
| 4 | 31/03/2017 | Friday | 11:00am | 1 hour | Lecture | 15 | WG403 |
| Week 6 | | | | | | | |
| 5 | 3/04/2017 | Monday | 9:00am | 1 hour | Lecture | 16 | WG403 |
| 6 | 3/04/2017 | Monday | 10:00am | 3 hours | Lab |  | WG403 |
| 7 | 6/04/2017 | Thursday | 8:00am | 1 hour | Lecture | 17 | WG403 |
| 8 | 7/04/2017 | Friday | 11:00am | 1 hour | Lecture | 18 | WG403 |
| Week 7 | | | | | | | |
| 9 | 10/04/2017 | Monday | 9:00am | 1 hour | Lecture | 19 | WG403 |
| 10 | 10/4/2017 | Monday | 10:00am | 3 hours | Lab |  | WT201 |
| 11 | 13/04/2017 | Thursday | 8:00am | 1 hour | Lecture | 20 | WG403 |
| MIDTERM BREAK – Fake hidden lecture – screwed up my note taking times. There is no listed lecture 22 in my journal | | | | | | | |
| 12 | 17/04/2017 | Monday | 11:00am | 9 hours | Self-Study |  | HOME |
| 13 | 19/04/2017 | Wednesday | 9:00pm | 8 hours | Self-Study |  | HOME |
| Week 8 | | | | | | | |
| 14 | 1/05/2017 | Monday | 9:00am | 1 hour | Lecture | 21 | WG403 |
| 15 | 1/05/2017 | Monday | 10:00am | 3 hours | Lab |  | WT201 |
| 16 | 4/05/2017 | Thursday | 8:00am | 1 hour | Lecture | 23 | WG403 |
| 17 | 5/05/2017 | Friday | 11:00am | 1 hour | Lecture | 24 | HOME (looked at slides) |

1. Lecture 13 – 27/03/2017
   1. **Overview:**
      1. Repetition (the for keyword)
      2. For Loops and arrays
      3. Common bugs when using for loops
      4. Coding standards
   2. **Lessons Learnt: For Loops**



* + 1. The syntax is that the for is the keyword, the int is the type of variable being counted. This could be a float that’s being counted down by 0.01 for example. The count is the name of the variable. The count > 0 part is the question or text being run, and the ++count is the next step down. This could be stepping down by 10, or growing exponentially as well.
    2. The counter doesn’t have to start at 0 either. This can start at any value or even a character like ‘Z’ and count down through the alphabet creating a quick 26 number countdown.
    3. The value that’s being set by the for loop can be used by anything INSIDE the scope. This means it can be used to find or set an index of an array, or set a timer, or be printed, or a lot of other things.



* + 1. Counting down, the variable needs to be set to the array size – 1. When you’re counting up, you need to make the count < [array size]. Exclusive.
    2. When going down, you must think of edge cases. So, if dimension is 6 then the int value should be one less so 5. And the rule should be k >= 0;
    3. When writing the command to happen each loop (the ++ or --), post and pre-fix are important. Post fix happens AFTER the assignment. Whilst pre-fix happens BEFORE the assignment.
    4. You can also write simple tests inside the for loop. Like my\_name[k] != ‘\0’ which waits until a null character.
    5. You can also ask multiple questions, or be logical when asking questions. && means asking first question AND second question. || means first question OR second question. ^^ means first question OR second question, but NOT both. And so on.
  1. **BUGS**



* + 1. The above code has syntax error. Where the end character has not been placed. This does not compile.



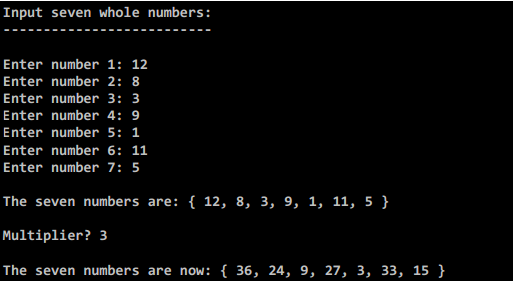
* + 1. The above code has an undeclared identifier. Where there has been no value set. You can move the value outside the loop by setting somewhere else, but you have to place the ‘;’ character to make the first section of the for loop question a NULL so it doesn’t make an error.
  1. **Coding standards**
     1. For loop is perfect for I you know how many increments you want. While loops are perfect for when you want to repeat a question to check for data.
     2. Writing for (; ;); runs a NULL loop and runs nothing forever. Because the NULL ‘question’ is evaluating to true, it runs forever.
     3. The computer tells you the above code would run forever.

1. Lab – 27/03/2017
   1. **Coding done:**
      1. In this lab, we went over and worked with the things from the week before. Repetition, if statements and conditional material.
      2. I worked on 7 different programs.
         1. Birthday season with switch was a program made so that when you typed in a number, it would print out what month your where linking and whether it was winter, or summer:

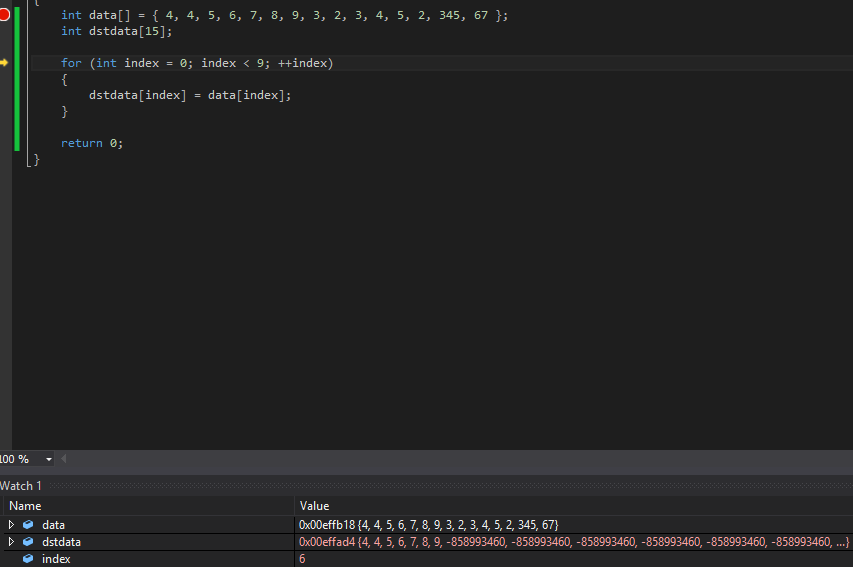
The program was quite simple and was just a switch case that looked at the month and under each case, printed something different for each month

* + - 1. Simple count up loop was an extremely basic count with a for loop, but instead of counting up by one, it counted up by 10.
      2. Stepping with a for loop was another simple program that did almost the exact same as the previous code, but it had three inputs. A starting number, a stopping number and a number to count up by. So the loop had to loop the amount of times as the third input (stepping size) and had to start at the first input, and stop at the second input (start/stopping numbers).
      3. Multiplication table was a simple program that asked for an input, and then have the entire times table from 0 to 14 of that input. The code had to loop a total of 15 times, and each loop, it multiplied your input by the index of the loop to get the multiplication table.
      4. Factorial calculator took an input and gave the factorial using a loop. Because of factorials being factorially big, or exponentially big in their answer, even with a long long variable type, the highest number that was possible to get a factorial of was 20 (answer is 2,432,902,

008,176,640,000) whilst the highest number possible on windows calculator is 1.973634253e+9997 or 19736342530000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

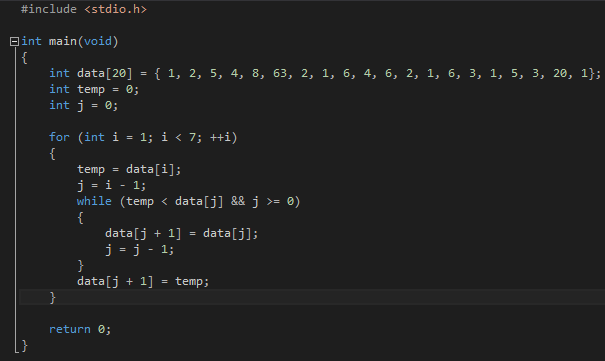
* + - 1. Loop and Array was made to take in 7 whole numbers and then asks to enter a multiplier. Then it prints out the list of numbers all multiplied by the multiplier. 
      2. The final one was ‘Your string length’ which asked for a word for you to input, and then printed out the string length of it.

1. Lecture 14 – 30/03/2017
   1. **Overview:**
      1. For loops and arrays
      2. Printing from an array with loops
      3. Copying between arrays with loops
      4. For loops with nested selection.
      5. Basic searching
      6. Criteria based searching
      7. Statistics computation
      8. Nested Loops: Loop within a loop.
      9. The goto word.
   2. **Lessons Learnt: Copying from array to array**
      1. If you set an array such as data [] = {1, 2, 4, 8, 16, 32, 64, 128, 256} you can then use loops to sort the data, search through the data or in this case. Cope to another array.
      2. To copy from array to array, set up a loop with it counting up to the size of the array, and then set the destination array with index of the int set from the loop, to the source array with index of the int set from the loop.

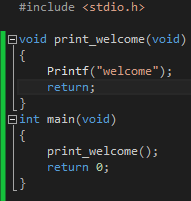


* + 1. The above code moves data from one array to another. If you want to copy data into an array backwards, you need to make the index for the source data the size – the index. (data[6 – index]) like so.
    2. You can also use loops to count how many letters are in a variable by making a loop that goes while the name != ‘\0’.

1. Lecture 15 – 31/03/2017
   1. **Overview: SCOPE**
      1. Local Variables.
      2. Variable lifespan
      3. Out of scope
      4. With sequence
      5. Selection
      6. Iteration
      7. Nesting.
      8. More nested loops.
   2. **Lessons Learnt:**
      1. How long does the variable exist? A variable exists within and only within its own scope. This also means variables are only accessible within the main if they’re set in main, and not in other functions. You can however, if you wanted to make a variable that existed and was accessible everywhere, set a variable outside all scopes. Bellow the #include statements.
      2. Compiler errors are quite common when trying to access or chance a variable that’s outside the scope. The compiler throughs an error showing that the variable is undefined because technically, the variable doesn’t exist anywhere but inside its own scope.
      3. The variable lifespan is only available with the scope of the block within the main for example, or in an if statement, or in a loop, ect….
      4. A sorting algorithm can set data within the loop, but it’s better to set variables outside loops. Like the given code bellow:

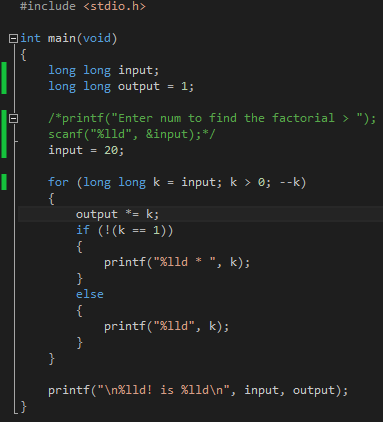


1. Lecture 16 – 3/04/2017
   1. **Overview:**
      1. Modularity
      2. Functions – Every C program written so far has at least one function. The Entry point, or main, is a function.
      3. Each function has a: Name, parameter (input data), and return type (output data).
      4. Int main(void)
         1. Name = main
         2. Input = none or void
         3. Int = the type of output data.
   2. **Lessons Learnt: Basics of a function**
      1. Invoking a function: To run a function, it must be *invoked* (also known as *called*).
      2. Calling functions is sometimes already set into the C compiler as a basic thing such as Scanf, printf, rand, srand, time, atoi, powf, sprint, ect…
      3. To call a function, you must specify the name of the function, and provide the required input. The () is how to call the function and there is the Call, Caller, Callee.
      4. Call:
         1. The invoke of the function
      5. Caller:
         1. The function that invoked the function currently running.
      6. Callee:
         1. The function being called.
      7. It is important to know that creating a function means that you’re leaving the main, so all data is not moved over. They are out of scope. To move data, you have to have it as an input type. The output type is the data given back from the function. In main, we use the return keyword. This is used to stop the execution of a function and return an output.
      8. There are different types of data that can be input or output. You can use every single type of normal data that is in C from ints, floats to enums and structures. One type of data IO value though that is most used is the Void type. The keyword void represents a type of data. But it stores no data. It can be used to indicate that a function takes no arguments. You can change the main from int to void. So that it passes nothing back. No data to be passed back.
      9. If no data is being passed back, you must change return 0; to return; so, it returns NULL.
   3. **Lessons Learnt: Creating our own functions**
      1. A function is like a modular program. A good function has a clear purpose, a name that reflects its purpose clearly. A good function conducts the actions that result in fulfilling an efficient purpose and they do a single job. Functions can have a lot in them, but they usually do one thing to be modular. Such as take in a bunch of variables and calculate floor functions and other calculations on them. Or a function could be used to calculate how much a cube is, ect…
      2. All of the above is why a good function is highly reusable.
   4. **Lessons learnt: Defining a function**
      1. Void print\_welcome (void)
      2. The above takes nothing in (void), when it’s done the job, it returns nothing cause of void. You can put one or more steps within that code. step in it.

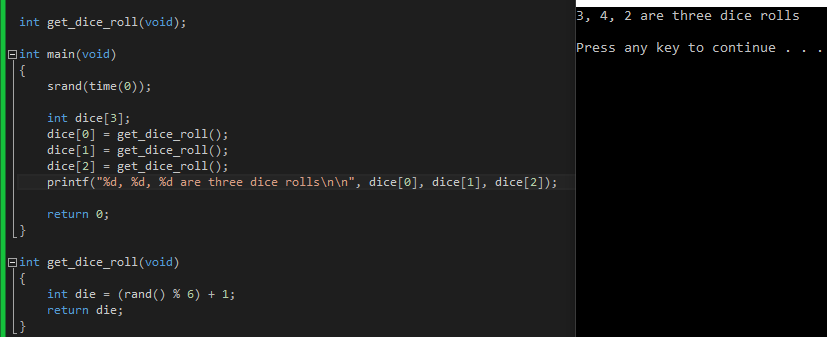


* + 1. The syntax of defining and declaring a function is as so: [output] [name] [input]. The order of function definitions matter:
       1. You have to define the function first before calling the function. This is specific so far to C (other languages might work differently).
       2. You must define the function OUTSIDE the int main(void) because that’s a function within a function.
    2. To properly set up a function, you have to first do a prototype. Which is to say, you are defining the function without instructions. Void print\_welcome(void);
    3. What this does is set up what the function will become and is set at the very beginning of your code. You can set the instructions later. Proper practise is setting the instructions after the int main(void). But setting the name, output and input at the start. Simply define it. This is analogous to a Black box: Function Declaration. You simply define it. You can write what it is and no need for instructions. You can write those later.
    4. In the compiler, a linker is something that finds where the instructions of a declared function is.
  1. **BUGS:**
     1. ERROR: LINK2019 – Unresolved external symbole\_print\_welcome referenced in function\_main. This means there is no instructions that are meant to be linked to a function you’ve called. The linker finds that you have a function defined, but it is not declared.
  2. **Lessons Learnt: The call Stack**
     1. The call stack stores all the information about the active functions of a program when executing. It’s sort of like saving live data into temp registers to be used straight after current program or function has finished running and has ‘popped’ off the stack.
     2. Things that are saved to the stack can range from input to the function or the data that will be returned from the function and variables that are declared in the function (local variables) …
     3. The call stack can be examined during debugging to see the information about and in the active functions.
     4. The stack frame works by creating something when it is called. When you call a function, in reality, it’s not sending anything to it, it’s ‘creating’ the function and running it in its own separate call stack plane, and when the function is done. The function is destroyed.

1. Lab – 27/03/2017
   1. **Coding Done/Lessons learnt:**
      1. When going through loops, it is important to remember the syntax and properly coding a loop. Using arrays with loops is very useful, however bugs can easily appear. I did not come across many bugs besides undeclared variables when I declared variables inside a scope and then tried using them outside the scope. This does not work. Programming a factorial calculator was the highlight of this lab as I had learnt about factorials ages ago and was excited to make a code that found one.
      2. Unfortunately, the factorial equation code, even when using a long long variable, cannot calculate above 20! Since there is an overflow and it becomes a negative value.

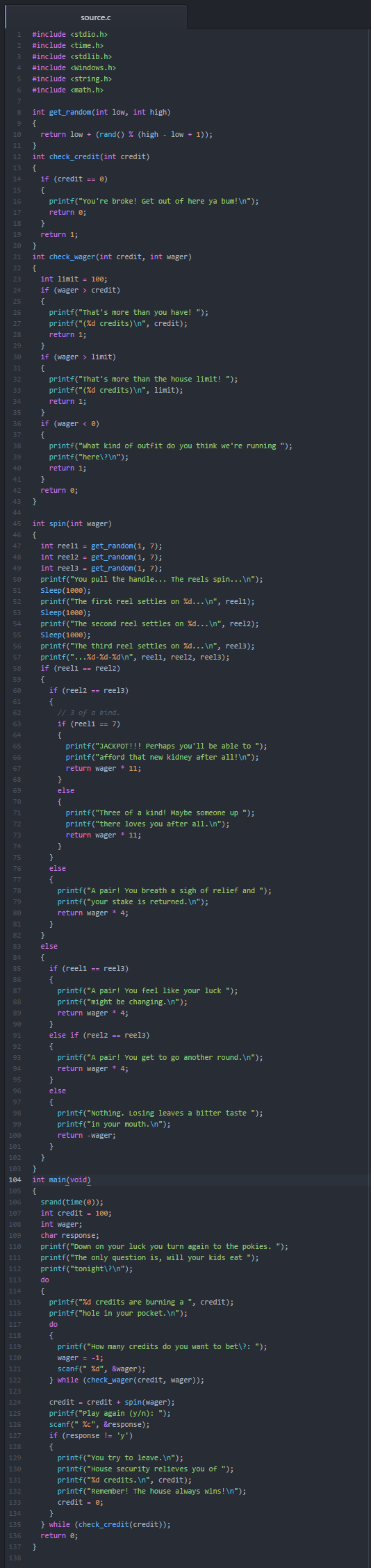


1. Lecture 17 – 6/04/2017
   1. **Overview:**
      1. Function calling with more in-depth building
      2. Using different forms of variables when calling functions
   2. **Lessons Learnt:**
      1. Calling a function can use different variables. Then function itself stores them into different variables. In essence, calling a function copies data and functions from the main, or any other callee, and sets them into a new place in RAM sometimes, under a new name.
      2. You can create variables with the function variables. Once a function is called with certain inputs, those inputs can be taken and modified or used. But they’re only copies of the originals. You can to re-set the data once the function has finished calling.
      3. You can also pass info back to the main using the return keyword. The return value can be anything as well, or nothing if you don’t want data passed back.

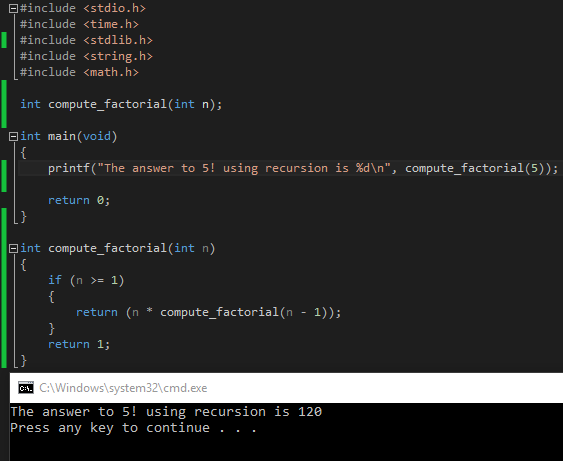


* + 1. The code above calls the function get dice roll three times, and each sets the output into an index in an array. This is really modular, and becomes almost, instead of a magic number needing a constant. This is where you have a magic block of code, so you make a constant block of code. That’s separate and unchanging and can be called as many times as needed.
    2. The return can also return two variables put together. Such as return(width \* height) which is passing back computation. You can also return the value of a variable as shown above.

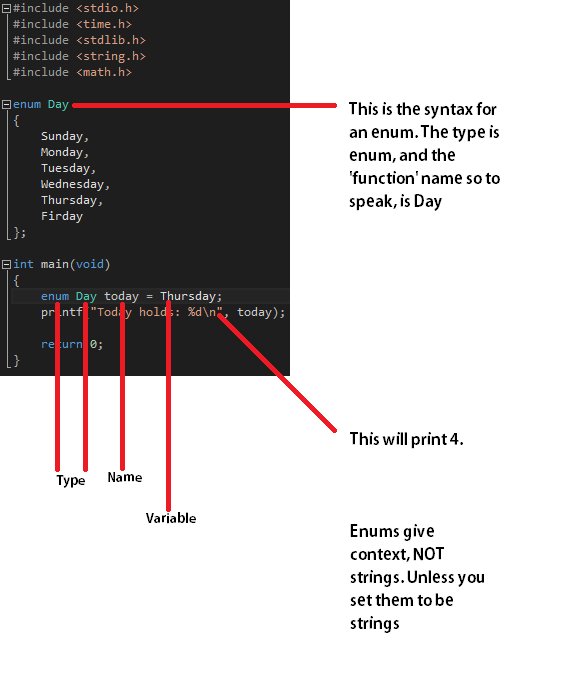
1. Lecture 18 – 7/04/2017
   1. **Overview:**
      1. Functions have IO, and local variables and its own scope.
      2. The class programmed a working slot machine. For this we started with the design, and then implementation.
      3. The function takes in the values and does computations with them.
   2. **Lesson: Programming a slot machine**
      1. It needs to store:
         1. Credits
         2. Players wager
         3. Results from reels
         4. Variables
      2. Functions need to be modular and need repetition selection.
      3. The requirements of this program are that if the player has money, they can place a bet. The player then spins the reels and the machine then spins. The machine then stops and shows its reels. If it’s triple numbers you win the jackpot (\*11 points). If it’s a pair, you get (\*4 points). If you win, you get paid, and if you lose, you lose credits.
   3. **The design Phase:**
      1. We need to know how do we program what we need. In order to know that, you must design what you have in mind. We also need to think of the tools we have to do so.
      2. The tools include known code.
      3. It is also important to think of what is known and what is unknown.
         1. The known for us is we can make variables, selection, create and use functions, print and scan data, and generate random numbers.
         2. The unknown for us is that we don’t know how to animate the effect of the machine reels moving for us. What we could use is a pause or sleep function.
      4. To get some sort of animation or to pause/sleep a program, we need to include the file windows.h This allows us to get the sleep function.
      5. The screenshot bellow is a 2560 by 1440 screen shot of the entire slot machine code.



1. Lecture 19 – 10/04/2017
   1. **Overview:**
      1. Functions; a Review. As well as looking at some common bugs with functions.
      2. Recursion. Stack overflow and base case.
      3. Enumeration: Declaring a new data type and using enumerations
   2. **Lessons Learnt: Declaring and defining**
      1. The difference between declaring and defining is that when you define, you’re setting up the function as it is, without any instructions. This is what the linker will find. This needs to be the same as the function when declaring. When declaring, you’re giving the function meaning and code to run.
      2. Functions must be named properly.
   3. **BUGS:** 
      1. Calling a function without braces (warning C4047)
      2. Not putting a return in. If it’s an int function, it has to have a return to pass back the data (warning C4716)
      3. Not saving the result of the function. When you run the function, it doesn’t automatically put it somewhere or save it or print out. You can and have to save or set the functions answer somewhere to save it.
      4. Not matching the declaring function to the defining function.
      5. Doing a NULL function when trying to define a function.
      6. Trying to call a variable out of scope.
   4. **Lessons Learnt: What happens when a function calls itself? This is recursion.**
      1. This is not like a loop, because a function is sent to the stack and has to be popped off, recursion will very quickly build upon the stack and crash the console and the program.
      2. An example of this is having an input and then having an if statement that asks if your input is the right number/answer. And then if it isn’t, it will recall the main. This is going to eventually finish but it has to pop off all the other called stacks. So it will save each stack so it will use more and more memory. A memory loop will occur. The compiler will through an error of stack overflow (c4717 warning)
   5. **Base Case (when should recursion stop and when should you use it?)**
      1. Using recursion well is very specific to the situation. It would work with factorial calculations as it loops down, and the stacks start disappearing, the factorial could be calculated.

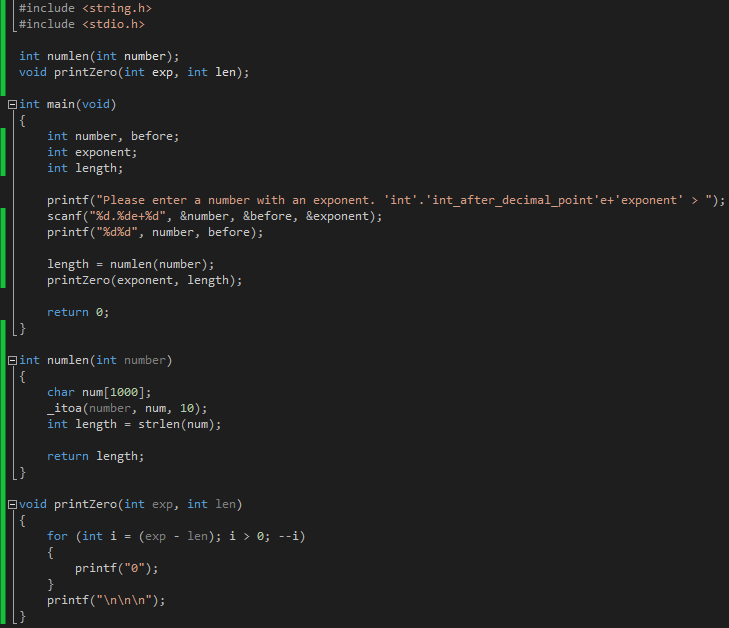


* 1. **Lessons Learnt: Enumeration**
     1. Creating a new data type. Enumeration allows you to make a new data type and use it similar to an array, but the values are set as constants.
     2. Different data types already exist. With ints being 4 billion different states, and chars having 256 states.
     3. You need to set info on what the numbers mean and what they represent. If you don’t, it could be random.
     4. For enums, they’re sort of like functions. You need to set them up first. Enums are also more of a visual fix to being confused by setting data. Instead of using day[4] which could really be anything, you can use day = Thursday if you’re using enums.
     5. To set an enum, you first need to declare it:



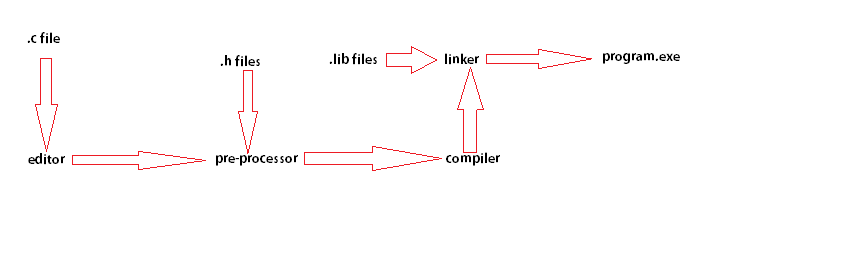
* + 1. You can also use them like arrays. Enums are like a list. You can set their definitions as well to be whatever you want. In the above code, inside the enum declaration, if I set Sunday to be 5, and Friday to be 10. Then the compiler will auto fill in the rest, and so Thursday will be equal to 7.
    2. Functions can also take in and pass out enums as well.
    3. Enums are like a list, you can easily add more and more loops that run to the end and it will still work.

1. Lab – 10/4/2017
   1. **Coding done:**
      1. The coding in this lab worked on the previous weeks learning with calling functions, testing scopes and returning data from functions. It also went over recursion a little and enumeration. The lab exercises where extremely easy and still quite basic, just incorporating functions instead of keeping everything within the main block like previous labs. Wasn’t super interested in making boring code, so I decided to make a program, that used functions to its advantage, to scan in an exponent number such as 1.245e+9 and print out the full version (124500000).

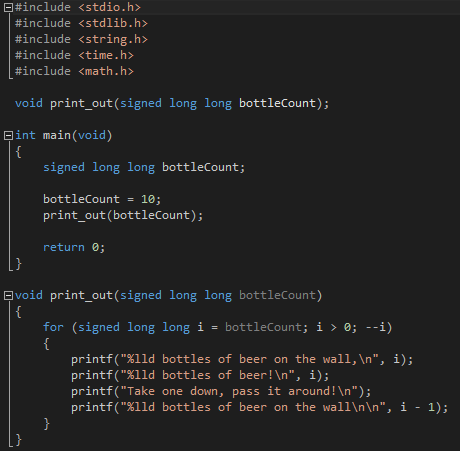


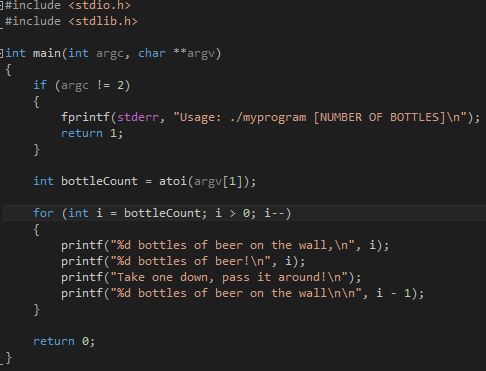
* + 1. I made a small function that works in the EXACT same way an strlen from string.h works, except for numbers because I wanted to count how long the number was. It then printed out the number before the decimal, then the numbers after that as one, then the proper amount of 0’s after it.

1. Lecture 20 - 13/04/2017
   1. **Overview:**
      1. Modularity: Functions and side effects.
      2. The c Tool Chain Revisited.
         1. The pre-processor. When it access header files such as .h files.
      3. Modularity:
         1. Interface vs implantation.
         2. Multi-file projects
      4. Static library files.
         1. .lib files. You can get colour text in console.
   2. **Lessons Learnt:**
      1. When you’re using a program, it changes state over time, this data is held in memory but changes over time.
      2. A functions job is to calculate a value or do a job or an action or do something specific. To add modularity to code so that little bits can be reused over and over with ease. Functions can’t be too specific to certain pieces of code.
   3. **Lessons Learnt: Header files and the Pre-processor**
      1. This is a build step that helps before compiling. #include at the top of a c file means that you’re including code from other files. In actuality, you’re breaking down what would normally be a massive amount of code within one file. Including stdio.h for example contains the definition and declaration for scanf, printf, and sprintf and fprintf. Each one of those is actually a function that is being fed extremely precise and dynamic information.
      2. A .h file is a header file, a .c is a source file.
      3. You can also do other things besides just #include. #define is similar to a macro. You can set a certain thing here. #undef removes a macro. #if controls conditional compilation, #elif, and #else both control conditional compilation as well. And #endif ends the first if statement.
      4. #ifdef checks if a token has been defined. Same as if defined. #ifndef checks if the token has NOT been defined.
      5. #include <stdio.h> the <> braces mean a system file.
      6. #include “plcolour.h” the “” mean local files.
   4. **Lessons Learnt: Define**
      1. #define creates a pre-processor macro. Defining for example, SIZE of 10 is an old way to set a constant value that can’t be changed. All definitions must be capitalised.
      2. #define RADTODEG(x) (x\*57.29578f). This code defines a token with the name RADTODEG that replaces any x with x \* 57.29578f which changes any radian value you enter as x, to degrees.
   5. **Lessons Learnt: IF DEF**
      1. #ifdef checks if a token is created, and then does a certain thing. This can be used extremely well when defining a certain console or platform when playing games. Like saying if you’re on Xbox or if you’re on PS4, or PC, then run other code than what you’re using already and so on.
      2. #ifdef EXAMPLE\_MACRO\_1 printf(“Stuff”);
      3. The code above checks if Example macro 1 is defined, and if it does, it prints stuff.
      4. This allows you to be extremely controlling with your code and be exclusive with how you code.
   6. **Lessons Learnt: Built in macros**
      1. There a bunch of built in macros that allow for more control over whatever you’re doing.
      2. --FILE--: this macro gives the name of your current source file.
      3. --LINE--: this macro gives the line number in the current file.
      4. --DATE--: This gives the date of the compilation of the file.
      5. --TIME--: This gives the compilation time of the file.
   7. **Static Library’s:**
      1. A static library is another part of the compiler and is a library that is shared publicly so as to keep it’s possibly expensive source code. You can also make your own static and header files.



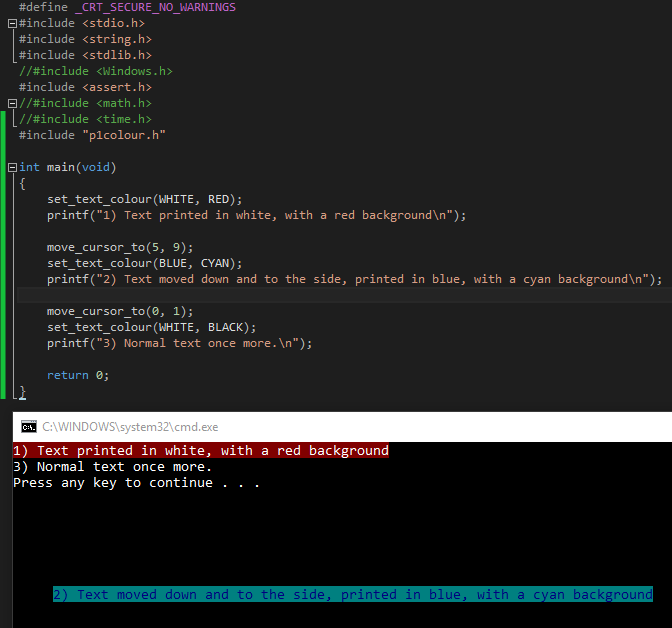
1. Self-Study – 17/04/2017
   1. **Code done:**
      1. I went through all the previous weeks code exercises that I didn’t get to, and I went over some other things before having some fun with random coding. I made a sorting program that’s similar to the sorting one we saw in an older lecture, except I made it more stream line and with less variables. Just setting a temp value, and that’s all. Then for fun, I tested around with setting compiler settings and getting the output of the console to a text file.
      2. I then decided to make a small program that printed out the bottles of beer on the wall song but starting from 1 trillion. I wanted to see how quickly that would take and how fast it would print out. What I didn’t expect was how big the file would get, and how quickly it would grow. The code bellow was tiny and did not take that long to make. A friend in a discord chat recommended for me to use the second images code, but I didn’t entirely understand it.

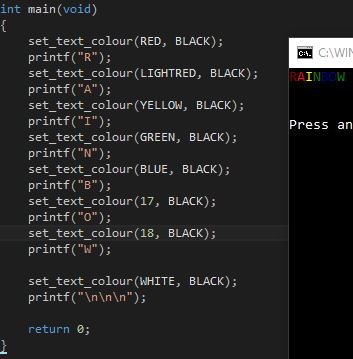
This is my code. I can replace the 10 with any number I want and it will print it out and save it into the text file.

This is the recommended code. I don’t know what fprintf does and I don’t know what the other things do, so I didn’t use it.

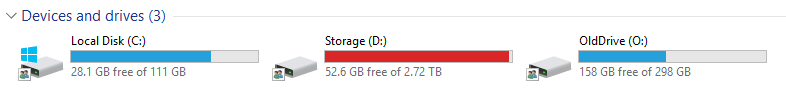
In the end, I set up my code with the bottleCount value set to 1 trillion and left for dinner, after coming back and seeing it was not done, I left it and did some other homework:

I worked on understanding p1colour.h to find out how to use it correctly. Adding p1colour to the program using #include is a way to both change colour of text and move the text around and start it at different positions. To move the ‘text’ you must use the function: move\_cursor\_to. Which takes in a column number and a row number. The actual function is declared inside p1colour.h and defined inside p1colour.lib so you can’t see what the actual code is. The function to set text colour is: set\_text\_colour and takes in two colours. A foreground and a background colour. Because the p1colour header file has an enum declared there, we can use normal words like WHITE and BLACK to get colours, otherwise, the functions take in normal numbers and use bit shifting and bit flags to combine the different values into one number. The code bellow has three sentances, one of which is placed out of sync and out of order using the move\_cursor\_to function. This means that the Press any key to continue…. Is placed in between where the text is normally. The text colour options are quite varied, but are missing orange and a few other colours that might not be available in the console.

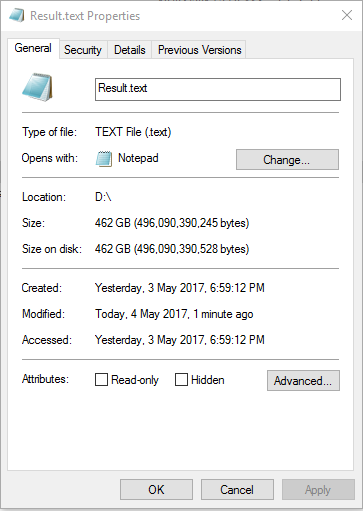


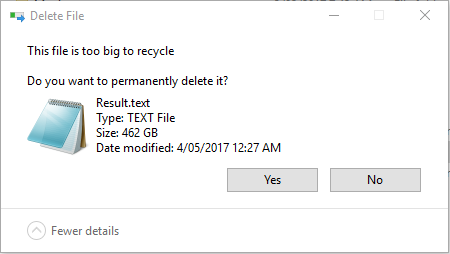
Going back to my original code after practising for a while with some more p1colour and creating one word that took on a different colour per character (didn’t work cause no orange, indigo or violet) 

As seen in the code to the left. I went back to the running program to see how big the file had gotten.



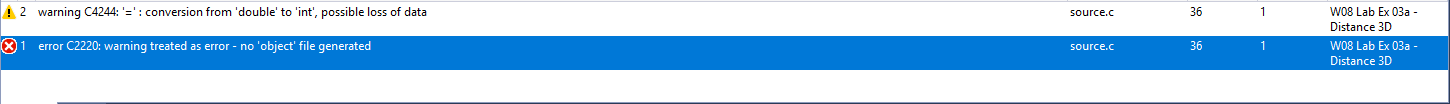
It had been around 1 and half hours, after playing around with the colour options I found it was still not done. I went to check some videos on my computer and saw that my file system was in the red and had around 200gb left. I was confused why it was red, as that usually meant it was getting full. So, I checked the files and found that the result text file being created had gotten to 200gb. After finding this, I decided to leave the program going for another 3 – 4 more hours while I worked on different codes and trying to find and use the actual declaration of set\_text\_colour. I ended up leaving the program going in the background to see how big it got, and I found it grew by about 1gb per second, and when I cancelled it. It was 480gb big.

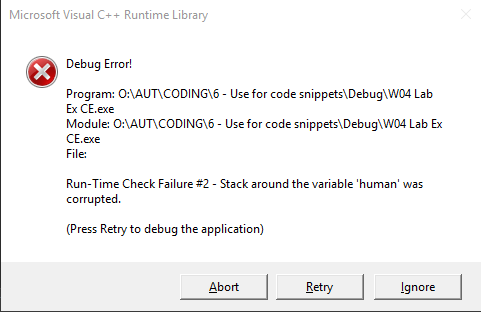




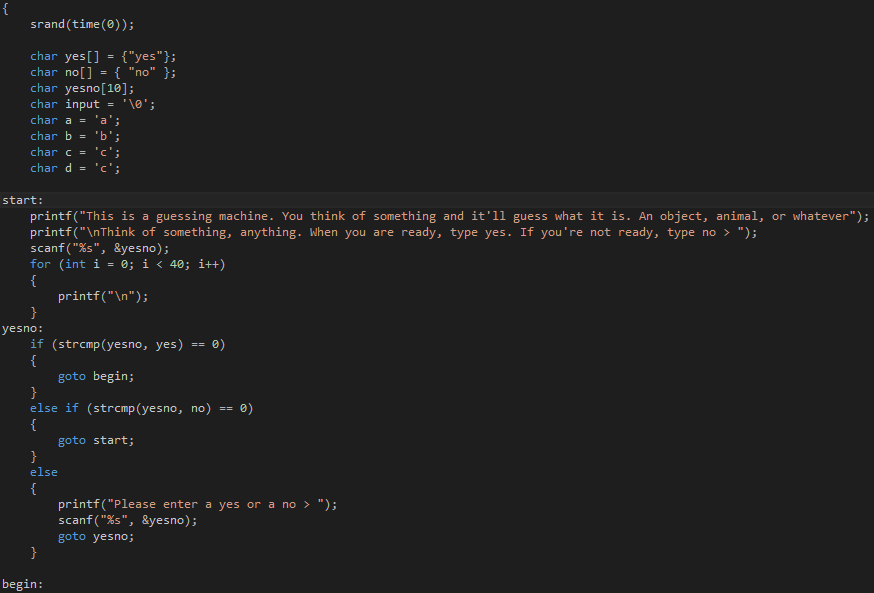
* + 1. **BUGS:**
       1. The bellow bugs where thrown and a lot of them where easily fixable, but some I found I couldn’t figure out without help from google.



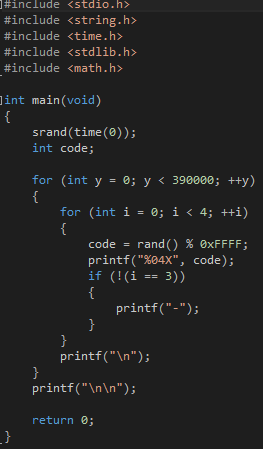




1. Self-Study - 19/04/2017
   1. **Code Done:**
      1. I finished programming all the exercises that I hadn’t finished the other day from self-study and starting working on testing with if in a wide range of things:

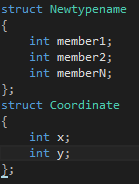


* + 1. I tried to make a guessing machine that tried to guess what object or whatever you were thinking of, but I found that depth of questioning would be to extremely high. I would have to go extremely deep with if statements to get all possible answers for what you’re trying to think of.
    2. I then worked on a brute force code generator to try and get a game code for steam, or another platform. I found that the chance of getting a code was extremely low, but I ended up creating some nested looping to create codes and then sending them to a text file.



* + 1. I originally started by trying to simply print random locations in the RAM and use those as codes because a RAM location is in the same format (almost) as a game code. But it wasn’t very malleable as a number to manipulate. I then tried generating one single hex code from 0, to FFFFFFFFFFFFFFFF (16 byte number) but I found that would be too big as well. In the end, I made a loop that printed the first, then second, then third, then fourth part of a single code and then doing that over by 100,000 times to generate 100 thousand randomly generated codes. Unfortunately, there are around 18 quintillion possible combinations, so the chance of using brute force to get a game code is extremely low.

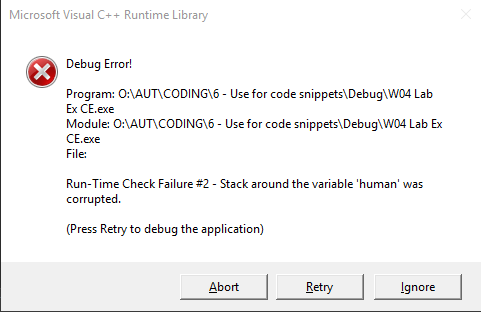
1. Lecture 22 - 1/05/2017
   1. **Overview:**
      1. Structures.
         1. The allow you to group several pieces of information together.
         2. They are a type for storing complex pieces of data.
         3. They are a collection different variables.
      2. Structures have members.
         1. These are the parts that make up a structure. They can also be called attributes, or fields or properties.
   2. **Lessons Learnt: Declaring a structure**
      1. Struct is the keyword to use.



* + 1. The coordinate structure is a structure made from two variables. It’s a complex group that is used as a blueprint for when you want to use it. You don’t save data into the declaration of a structure. It’s use to declare the blueprint for when you set a coordinate struct later, and where you meant to use data and how you’re meant to use it.
  1. **Lessons Learnt: Designing a structure**
     1. You need cohesion. The degree which elements belong together.
        1. High cohesion means there is robustness, reliability, reusability and understandability.
        2. Low cohesion means it’s difficult to maintain, test, reuse or understand.
     2. If you’re making a structure for a rectangle, then to make it understandable and reliability, you should name the members height, width, and maybe colour or whatever. If you’re making a structure for a 3D vector (a point on a graph), then naming member as x, y, and z is cohesive.
     3. Structures are reminiscent of an enum, but are MUCH more reliable, much easier to understand, much more robust and way more reusable. They can hold both text, numbers or floats.
     4. To use them, it’s like using enums. Setting a [type] [name] = something. The type is the struct Coordinate for example.
     5. To access the members within a structure, you have to use the dot operator.
        1. Steffan\_Hooper.age = 65;
        2. Steffan\_Hooper.height = 100;
        3. Steffan\_Hooper.weight = 300;
     6. The dot operator allows you to select a single member from a structure. You can also use strcpy from string.h library to put strings into the members.
  2. **Lessons Learnt:**
     1. Structure variables can be passed into functions as input. You can also use sizeof() to find the size of a structure.
     2. Structures take up memory. Each member takes up a certain space in a certain location in the RAM.
     3. You can also make a member an array, allowing for something like Steffan\_Hooper.poopTime[500657] = 1200. AND, you can make a structure array.
     4. If you go:

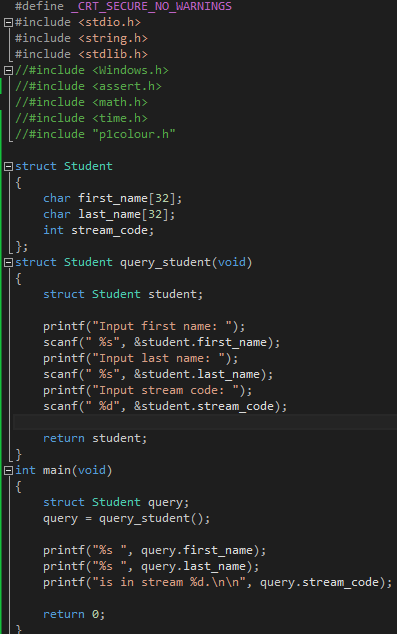


* + 1. You can set multiple parts of an array. And if you wanted, you could set the structure of human to an array and have multiple people stored inside that file. Except the code above kept through this error on my laptop when I was playing around with this code for about 2 hours after the lab.

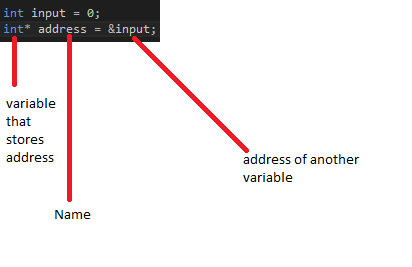


* + 1. I never ended up having time to fix it unfortunately.

1. Lab – 1/05/2017
   1. **Coding Done**
      1. This lab was working with structures and members within structures and getting them to work together and being able to set data into structures as well as passing them into functions.
      2. The first code was called cat structure and was used to store the details of a cat to be printed later. The cats had an age, weight, tail length and name. The age, weight and tail length where all integers, and the name was a char text array. The car structure was declared in main and then data was set into it. The age, weight and tail length where just cat.age = 13 for example. Where data could be directly set into the variable. Whilst the name needed to use a sprintf to print data into the variable.
      3. The second exercise was called structure as a parameter and was similar to the cat structure except it was for a soft drink. It contained a name, serving size, energy content, caffeine content and max daily intake. The name was a char text array, the serving size, energy content and max daily intake where all integers and the caffeine content was a floating-point variable.
      4. The third exercise was called distance 3D and was about 3D vectors. To calculate the distance between two different 3D vectors, the formula is distance = the square root of (x2 – x1)2 + (y2 – y1)2 + (z2 – z1)2. To do so, an array of structures was made. Each structure contained an x, y and z vector and there were two elements inside the structure array. One each for each 3D vector. Then the formula was applied and saved to a separate variable and printed out using a function. The two structures where passed into the function and calculated to get the distance.
      5. The fourth exercise was to create a structure and then take input and save into the structure but have the saving part in a function separate to the main. This was quite simple and just used scanf to save into each member. This was the last exercise I got up to:



1. Lecture 23 – 4/05/2017
   1. **Overview:**
      1. Addresses
      2. Pointers
         1. Dereferencing
         2. Null pointers
         3. Wild pointers
         4. Dangling pointers
      3. Basic pass by reference
         1. Returning multiple things
      4. Coding standards: Pointers
   2. **Lessons Learnt: Addresses:**
      1. & means saving to an address. This is an address. When you use this to scanf, you are essentially saying please save the incoming input to the address of this variable.
      2. The addresses are shown as a long hexadecimal number like 0xFFFFFFFF which is a 32-bit number.
      3. You cannot save or write to the address of 0x0000000 or 0 because that is a special address with contains information about start up and bios.
      4. %x means you’re writing in hex, %X is capital hex. And %p is printing out the address of something. To print an address of something, you must give the printf the address of a variable and use the %p command. %p is the same essentially of printing in hex, but filling in all the empty spaces with 0’s.
   3. **Lessons Learnt: Pointers:**
      1. The \* symbol has a lot of meanings. It can also be used as a pointer. A pointer is a variable that saves the address of another variable and can be deferenced and ‘undone’ to write data to the original variable THROUGH the pointer.

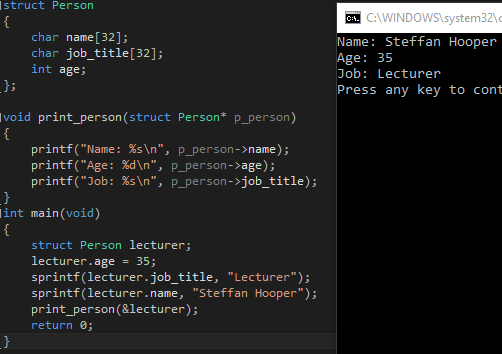


* + 1. When you scanf into a pointer, you only need to put the name of the pointer and the not the address ‘ampersand’ because it is then saving into the original variable.
    2. The syntax for pointers is as shown above. The type is an int\* and then you set the name, and then you set what the pointer is pointing at.
    3. Pointers can be any type of variable. An int, char, float or double. But it has to be a pointer to the same type of data.
  1. **Lessons Learnt: Deferencing:**
     1. The \* symbol has another meaning and that is to deference. A pointer takes in an address and you can then deference it to go to the place in memory from the pointer and then print it:

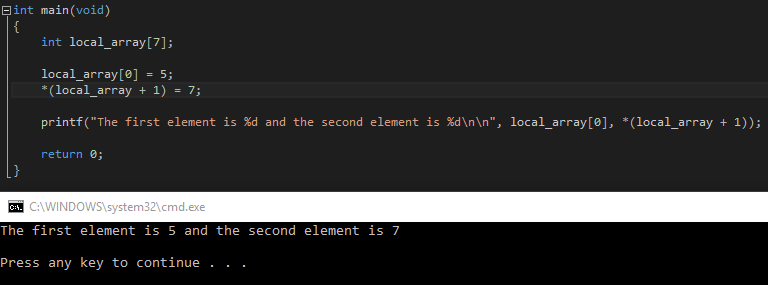
Printf (“address points to: %d\n”, \*address);

* + 1. The above code basically undoes the pointer and goes straight to the original variable and prints that. Alternatively, printing out the address pointer without deferencing also does the same thing.
  1. **Lessons Learnt: Pointer Terminology:**
     1. NULL Pointer
        1. A NULL pointer is a pointer to an address that is currently empty or unknown. This is extremely useful later as if a function needs to take in a pointer, but the function doesn’t know the data type, it can take in a void pointer and then you can use type casting later to allow the program to understand what it is taking in. A NULL pointer is essentially a backup pointer that points to nothing currently but does hold unknown data.
        2. A Wild Pointer is a pointer to a random location. If a pointer is set up but doesn’t point to anything, then it could point to anywhere in memory. Sometimes this is not worrisome, but sometimes this is extremely bad as it can point to dangerous places in RAM that could cause crashes. A wild pointer is EXTREMELY bad and unpredictable. It is much better to make a NULL pointer first rather than make a wild pointer if you do not know yet what you want the pointer to point at.
        3. Dangling pointer: Once held a valid point, but it went out of scope and it becomes invalid. This is not possible without the program crashing with warning C4142. If a variable is set inside an if statement, and then the pointer leaves the if statement but the variable doesn’t, then it becomes a dangling pointer.
     2. **Lessons Learnt: Pass by reference:**
        1. “Don’t take a copy of memory in a function, take its address. And then deference it.” This actually allows functions to access out of scope data and change it. Anything and anything can be passed by reference and because functions are so powerful, and can take in structures and arrays and such. Passing by reference is extremely powerful.

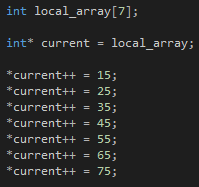
1. Lecture 24 – 5/05/2017
   1. **Overview:**
      1. Pointers to structures
         1. The -> operator
         2. Passing structures by reference
      2. Arrays and pointers
         1. Passing arrays – pass by reference
         2. Passing arrays as pointers
      3. Pointer arithmetic
      4. C-String and pointers
      5. Pointers to pointers
   2. **Lessons learnt: pointers to structures**
      1. It is possible to get the address of a structure variable instance. To do so, you must first declare a structure variable instance and set its members and then print the address of the structure variable instance named ‘*something*’. This means it is possible to save the address of a structure variable instance into a pointer.
      2. The -> operator allows access to the members of a structure via pointer. If we set a pointer to a structure, you can then access individual members by going [*structure*] -> [*Member*] and setting the member from there. This also works with functions when passing a structure pointer into a function. If you were to need the use of a dot operator for some reason, you would need to first deference the pointer.
      3. Structures can be passed to functions via pointer. This allows the callee to have access to the caller’s data.



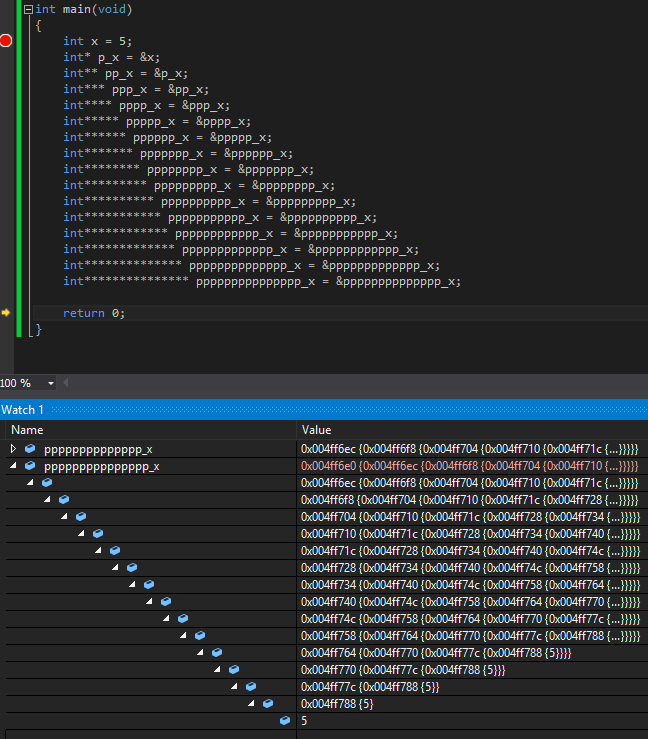
* 1. **Lessons learnt: Passing arrays as pointers:**
     1. IMPORTANT: When saving an array as a pointer, the pointer is actually the pointer to the first element of the array. When passing an array into a function, you essentially passing the first element into the function and allowing the function access to the address of the rest of the array.
     2. However, it is important to note that when accessing a pointer and treating the data as an array the programmer must be certain that the pointer really does refer to an array and that the data does not overflow the array. Often when passing by reference with arrays, a second variable is given to determine the size so as to never go over the size of the array.
  2. **Lessons Learnt: Pointer arithmetic:**
     1. When you access an array, element using the [] bracket notation, it’s actually using pointers to point to the address of elements inside the array. Multidimensional arrays are essentially pointers to pointers to pointers per element. Because of this, we can use pointer arithmetic to move along the pointers to get the next address and so on:



* + 1. When adding to, or subtracting from a pointer the sizeof() function to get the element pointed to will be used.



* + 1. In the code above, the address of the first element in the array is stored in *current* and then as the elements in the array are assigned into the array, the current pointer is incremented which moves through the array one element at a time.
    2. Using pointer arithmetic and pointer logic, saying a char text array like char array[] = “Stuff” is the same as saying you’re making a pointer to a char. Like char\* array = “Stuff”.
  1. **Lessons Learnt: Pointers to pointers:**
     1. This is possible as a pointer to a pointer is essentially a pointer to an address which holds a pointer to a variable. You can do this infinitely.



The code above makes a pointer to a pointer and so on, 15 levels deep to an original number.



The code right here is how you would print a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to a pointer to an integer.

Bibliography:

Abbott, J. (2017). *Weekly Labs* [Course Exercises and PowerPoint slides]. Learning Basic C. Retrieved from:

<https://blackboard.aut.ac.nz/webapps/blackboard/content/listContent.jsp?course_id=_85303_1&content_id=_3944184_1&mode=reset>

Burgener, R. (1988) *20Q, Twenty Questions*. 20q.net, Wikipedia. Retrieved from: <http://www.20q.net/>

<https://en.wikipedia.org/wiki/20Q>

Donati, L. (2013). *how to represent number bigger than long in c [duplicate].* Stack Overflow. Retrieved from:

<http://stackoverflow.com/questions/18675101/how-to-represent-number-bigger-than-long-in-c>

Hooper, S. (2017). *Weekly Lectures* [PowerPoint slides]. Learning Basic C. Retrieved from:

<https://blackboard.aut.ac.nz/webapps/blackboard/content/listContent.jsp?course_id=_85303_1&content_id=_3944184_1&mode=reset>

Hooper, S., Abbott, J. (2017). *Coding Examples* [Course Related coding examples]. Learning Basic C. Retrieved

from: [https://blackboard.aut.ac.nz/webapps/blackboard/content/listContent.jsp?course\_id=\_85303  
\_1&content\_id=\_3944184\_1&mode=reset](https://blackboard.aut.ac.nz/webapps/blackboard/content/listContent.jsp?course_id=_85303_1&content_id=_3944184_1&mode=reset)

Scott, T. (2014). *How the Self-Retweeting Tweet Worked: Cross-Site Scripting (XSS) and Twitter*. YouTube.

Retrieved from: <https://www.youtube.com/watch?v=zv0kZKC6GAM>

